|  |
| --- |
| El lenguaje ensamblador |
| Fundamentos |
| Silvina Monteleone, Eliana Ríos Leticia Quiroga |

[Introducción 3](#_Toc328407903)

[Programando 3](#_Toc328407904)

[Diferenciando interpretes, compiladores y ensambladores 4](#_Toc328407905)

[Ensambladores y macroensambladores. 4](#_Toc328407906)

[Ejemplos de ensambladores y macroensambladores: 5](#_Toc328407907)

[Proceso de ensamblado 5](#_Toc328407908)

[Proceso de liga (link), las rutinas de run-time y los servicios del sistema operativo. 6](#_Toc328407909)

[El sistema operativo (Linker) 6](#_Toc328407910)

[El formato del ensamblador 7](#_Toc328407911)

[Estructura 7](#_Toc328407912)

[Irrupciones 8](#_Toc328407913)

[Conclusiones 10](#_Toc328407914)

[Actividades 11](#_Toc328407915)

[Bibliografía 12](#_Toc328407916)

# Introducción

Cada procesador posee un lenguaje ‘maquina’. Este lenguaje es el nivel más bajo y posee una serie de instrucciones que solo pueden ser reconocidas y ejecutadas por el microprocesador. El lenguaje es un conjunto de números representando operaciones que ejecuta el microprocesador a través de su circuito interno. O sea, el lenguaje ensamblador es aquel que por cada uno de sus enunciados se corresponde una instrucción en la máquina.

PORQUE SE PROGRAMA A NIVEL MÁQUINA?

Porque permite interactuar directamente con el microprocesador, lo cual lo hace más rápido. Además, los programas de ensamblador ocupan mucho menos espacio (no tiene librerías y adicionales que si poseen los lenguajes de alto nivel). Es más flexible por la poca limitación que existe en el uso de recursos de la máquina. Los lenguajes de alto nivel poseen ciertas restricciones que limitan el desarrollo de ciertas aplicaciones.

Ahora, estas ventajas conllevan programar con cierta complejidad, ya sea por la complejidad de la programación en si y además porque la posibilidad de cometer errores graves en la lógica es alta.

No nos olvidemos de la portabilidad, cada máquina tiene su ensamblador, y a menos que dos procesadores hayan sido diseñados para ser compatibles, cada ensamblador tendrá sus especificaciones características.

Ej.: 80486 ejecuta su propio código y también los de versiones anteriores (de la misma serie) estos son desde un 8086 hasta un 80386. Un PowerPC es capaz de ejecutar instrucciones de los microprocesadores Motorola 68xxx y de los Intel 80xx/80x86. En el primer caso se siguió un estándar de compatibilidad entre modelos, en el segundo se extendió a otra marca.

Sin embargo, un microprocesador no podrá ejecutar un código de un modelo superior, ya que este posee modificaciones provenientes de mejoras que implican un cambio en la tecnología, que no fueron contemplados en la versión anterior.

# Programando

Si bien con el lenguaje ensamblador poseemos poder total sobre el procesador, este conlleva grandes riesgos y una dificultad alta para especializarse. No tanto por su sintaxis compleja, sino porque se debe calcular y trabajar con las direcciones de memoria de los datos, los saltos y las direcciones de llamadas a subrutinas, además de que para poder hacer ejecutable un programa, se deben enlazar las rutinas de *run-time* y servicios del sistema operativo. Este proceso es al que se le denomina *ensamblado de código*. Para facilitar la elaboración de programas se crearon los **Ensambladores** y el **Lenguaje Ensamblador**.

Cada uno de los valores numéricos del lenguaje de máquina tiene una representación simbólica de 3 a 5 letras en ensamblador (como instrucción del lenguaje). También, este lenguaje posee un grupo de *pseudo-operaciones* (también conocidas como *directivas* del ensamblador) que sirven para definir datos, rutinas y todo tipo de información para que el programa ejecutable sea creado de determinada forma y en determinado lugar.

# Diferenciando interpretes, compiladores y ensambladores

Aunque el lenguaje ensamblador haya sido creado para facilitar la programación del microprocesador, este es bastante complejo, lo cual hace que no haya muchos expertos disponibles. Para facilitar aun más la tarea de los programadores se crearon los lenguajes de alto nivel, que minimizan significativamente las instrucciones. Cada una de los enunciados de un lenguaje de alto nivel equivale a varias instrucciones de ensamblador.

Hay programas diseñados para traducir las instrucciones del nivel superior al más bajo. Los intérpretes al leer una instrucción, van buscando y ejecutando el equivalente de la instrucción recibida en acciones. Cada vez que se encuentra una instrucción, se llama una determinada rutina de lenguaje de máquina que se encarga de realizar las operaciones asociadas, pero en ningún momento se genera un código objeto y mucho menos un código ejecutable.

A diferencia de los compiladores, que en vez de llamar y ejecutar una rutina en lenguaje de máquina, éstos juntan esas rutinas para formar el código objeto que después de enlazar las rutinas de run-time y llamadas a otros programas y servicios del sistema operativo, da como output en el programa ejecutable.

Finalmente, tenemos los ensambladores que lo único que hacen es cambiar toda referencia simbólica por la dirección correspondiente, calcular los saltos, resolver referencias y llamadas a otros programas, y realizar el proceso de enlace. Los ensambladores son programas destinados a realizar el ensamblado de un determinado código.

**Ejemplos de interpretes:** DBase, BASIC, APL, y Lisp

**Ejemplos de compiladores:** Fortran, Clipper, COBOL, Pascal o C

**Ejemplos de ensambladores:** M[IPS](http://es.wikipedia.org/wiki/MIPS), Sun [SPARC](http://es.wikipedia.org/wiki/SPARC), HP [PA-RISC](http://es.wikipedia.org/wiki/PA-RISC), [x86](http://es.wikipedia.org/wiki/X86), etc

# Ensambladores y macroensambladores.

Si bien existen diferentes ensambladores, la diferencia entre los ensambladores radica en la forma de generar el código y en las directivas con que cuente, aunque estas diferencias son mínimas. El código ensamblador no cambia puesto que los microprocesadores con los que se va a trabajar son comunes. Así, todos los programas que se creen con un ensamblador en particular podrán ser ensamblados en otro, cambiando las pseudo-operaciones no reconocidas por el equivalente indicado en el manual de referencia del paquete empleado.

El término *macroensamblador* es usado para indicar que el ensamblador en cuestión tiene la capacidad de poder ensamblar programas con facilidad de *macro*. Una macro es una pseudo-instrucción que define un conjunto de instrucciones asociadas a un nombre simbólico. Por cada ocurrencia en el código de esta macro, el ensamblador se encarga de substituir esa llamada por **todas** las instrucciones asociadas y, en caso de existir, se dejan los parámetros con los que se estaba llamando la macro y no con los que había sido definida. Es importante señalar que no se deja una llamada, como a una subrutina o procedimiento, sino que se incorporan todas las instrucciones que definen a la macro.

# Ejemplos de ensambladores y macroensambladores:

**Macro Ensamblador IBM**.- Está integrado por un ensamblador y un macroensamblador. En gran medida su funcionamiento y forma de invocarlo es sumamente similar al de Microsoft. Su forma de uso consiste en generar un archivo fuente en código ASCII, se procede a generar un programa objeto que es ligado y se genera un programa .EXE.

**Macro Ensamblador de Microsoft**.- Dependiendo de la versión, este ensamblador es capaz de soportar el juego de instrucciones de distintos tipos de microprocesadores Intel de la serie 80xx/80x86. En su versión 4.0 este soporta desde el 8086 al 80286 y los coprocesadores 8087 y 80287. Requiere 128KB de memoria y sistema operativo MS-DOS v2.0 o superior. Trabaja con un archivo de código fuente creado a partir de un editor y grabado en formato ASCII. Este archivo es usado para el proceso de ensamble y generación de código objeto. Posteriormente, y con un ligador, es creado el código ejecutable en formato .EXE.

**Turbo Editassm**.- Este es desarrollado por Speddware, Inc., y consiste de un ambiente integrado que incluye un editor y utilerías para el proceso de ensamble y depuración. Es capaz de realizar el ensamble línea a línea, conforme se introducen los mnemónicos, y permite revisar listas de referencias cruzadas y contenido de los registros. Este ensamblador trabaja con tablas en memoria, por lo que la generación del código ejecutable no implica la invocación explícita del ligador por parte del programador.

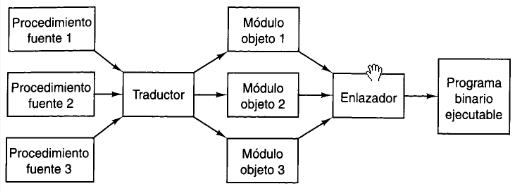
**Turbo Assembler**.- De Borland Intl., es muy superior al Turbo Editassm. Trabaja de la misma forma, pero proporciona una interfaz mucho más fácil de usar y un mayor conjunto de utilerías y servicios.

# Proceso de ensamblado

El ensamblado posee dos ‘pasadas’: En la primera se crea la Tabla de Símbolos, basándose en lo leído en el programa origen. Generalmente se crean tres: una de símbolos, una de pseudoinstrucciones y una de códigos de operación.

En la segunda pasada se genera el programa objeto y se crea un listado de ensamblado. Además de recolectar cierta información para el enlazador (LINKER)

Luego del ensamblaje ocurre el enlazado, que se hace antes de la ejecución para chequear dependencias a otros programas.



Durante la segunda pasada se crea el archivo objeto (ensamblado) referenciando la tabla creada en la primera pasada.

Todos los enunciados del programa origen se procesan antes de leer el siguiente. El procesamiento incluye la examinación de los campos etiqueta, código de operación y operando. La tabla de códigos de operación se revisa para encontrar un código operacional similar.

Si un mnemónico corresponde a una operación normal, el código maquina normal se inserta al archivo objeto. Si hubiese un error detectado por el ensamblador, se muestra antes de la línea que contiene dicho error.

# Proceso de liga (link), las rutinas de run-time y los servicios del sistema operativo.

Para poder crear un programa ejecutable partiendo de un código objeto se requieren llamadas a otros programas y a los servicios del sistema operativo, y agregar las rutinas o información de *run-time* para que el programa pueda ser cargado a memoria y ejecutado. Este proceso es lo que se conoce como *Link* o *proceso de liga,* y se realiza a través de un *ligador* o ***Linker*** que toma de entrada el **código objeto** y produce de salida el **código ejecutable**.

Estas rutinas de run-time sirven para que el sistema operativo pueda tener control sobre el programa en cualquier momento, ya que la asignación de recursos y su acceso deben hacerse solamente a través del sistema operativo.

# El sistema operativo (Linker)

Uno de los componentes que caracterizan los computadores personales es su sistema operativo. Una PC puede correr varios sistemas operativos: CP/M, CP/M-86, XENIX, Windows, PC-DOS, y MS-DOS. Lo que los define es la forma en que están integrados sus servicios y la forma en que se accede a ellos. Esto es precisamente lo que el *linker* debe enlazar y resolver.

Ejemplo con MS-DOS, (versiones 3.0 y superiores). Este sistema operativo está organizado de la siguiente manera:

|  |  |
| --- | --- |
| Comandos Internos | (Reconocidos y ejecutados por el COMMAND.COM) |
| Comandos Externos | (.EXEs y .COMs ) |
| Utilerías y drivers | (Programas de administración del sistema) |
| Shell | (Interfaz amigable, sólo versiones 4.0 o mayores) |
| Servicios | (Interrupciones) |

Servicios:

Más conocidos como *interrupciones* o *vectores de interrupción*, es parte mas fundamental de lo que es MS-DOS, y son mas que rutinas definidas por MS-DOS y el BIOS, ubicadas a partir de una localidad de memoria específica. La manera de acceder a estas rutinas y a los servicios que ofrecen es mediante una instrucción que permite ejecutar una interrupción.

MS-DOS proporciona dos módulos: **IBMBIO.COM** (provee una interfaz de bajo nivel para el BIOS) e **IBMDOS.COM** (contiene un manejador de archivos y servicios para manejo de registros). El acceso a los servicios del computador se realiza de la siguiente manera:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Programa de usuario** |  | **DOS Alto nivel** |  | **DOS Bajo nivel** |  | **ROM** |  | **EXTERNO** |
| Petición de I/O | ‹—› | IBMDOS.COM | ‹—› | IBMBIO.COM | ‹—› | BIOS | ‹—› | Dispositivo |

# El formato del ensamblador

|  |  |
| --- | --- |
| **Notación** | **Significado** |
| **Negritas** | Comandos, símbolos y parámetros a ser usados como se muestra. |
| *Itálicas* | Todo aquello que debe ser reemplazado por el usuario |
| **** | Indican un parámetro opcional |
| ,,, | Denota un parámetros que puede repetirse varias veces |
| ¦ | Separa dos valores mutuamente excluyentes |
| letra chica | Usada para ejemplos. Código y lo que aparece en pantalla |

Cada programa en lenguaje ensamblador es creado a partir de un *archivo fuente* de código ensamblador. Estos son archivos de texto que contienen todas las declaraciones de datos e instrucciones que componen al programa y son agrupados en áreas o secciones, cada una con un propósito especial. Las sentencias en ensamblador tienen la siguiente sintaxis:

**[***nombre***]** *mnemónico* **[***operandos***] [*;****comentarios***]**

# Estructura

Todos los archivos fuente tienen la misma forma: cero o más segmentos de programa seguidos por una directiva **END**. No hay una regla sobre la estructura o el orden que deben seguir las diversas secciones o áreas en la creación del código fuente de un programa en ensamblador. Sin embargo la mayoría de los programas tiene:

* segmento de datos,
* un segmento de código
* un segmento de stack,

Estos pueden ser puestos en cualquier lugar.

Definición de datos y declaración de instrucciones y operandos:

Reconoce el conjunto de caracteres formado por letras mayúsculas, letras minúsculas (excluyendo caracteres acentuados, ñ, Ñ), números, y los símbolos: ? @ \_ $ : . [ ] ( ) ‹ › { } + - / \* & % ! ´ ~ ¦ \ = # ˆ ; , " ‘

Declaración de números:

Un entero se refiere a una combinación de dígitoshexadecimales, octales, decimales o binarios, más una raíz opcional. La raíz se especifica con **B**, **Q** u **O**, **D**, o **H**. El ensamblador usará siempre la raíz decimal por defecto, si se omite la especificación de la raíz (la cual se puede cambiar con la directiva **.RADIX**). Así nosotros podemos especificar un entero de la siguiente manera: *dígitos*, *dígitos***B**, *dígitos***Q**  o *dígitos***O**, *dígitos***D**, *dígitos***H**. Si una D o B aparecen al final de un número, éstas siempre se considerarán un indicador de raíz, e.g. 11B será tratado como 112 (210), mientras que si se trata del número 11B16 debe introducirse como 11Bh.

Declaración de números reales:

Se designa con la **R**, que sólo puede ser usado con números hexadecimales de 8, 16, ó 20 dígitos de la forma *dígitos***R**. También puede usarse una de las directivas **DD**, **DQ**, y **DT** con el formato **[**+¦-**]***dígitos****.****dígitos***[E[**+¦-**]***igitos***]**.

Cadenas de carácter y constantes alfanuméricas:

Son formadas como **´***caracteres***´**  o **"***caracteres***".** Para referencias simbólicas se utilizan cadenas especiales denominadas *nombres*.

*Nombres:*

son cadenas de caracteres que no se entrecomillan y que deben comenzar con una **A..Z** **¦ a..z** ¦ **\_ ¦** $ **¦** @los caracteres restantes pueden ser cualquiera de los permitidos, y solamente los 31 primeros caracteres son reconocidos.

# Irrupciones

La mayoría de las rutinas que controlan al computador están grabadas en el ROM del BIOS, aunque muchas rutinas son establecidas por el sistema operativo y se cargan en RAM al momento de encender al computador. Estas rutinas son denominadas *interrupciones* y son activadas mediante la instrucción: **INT** *número*. Una *interrupción* es una operación que invoca la ejecución de una rutina específica que suspende la ejecución del programa que la llamó, de tal manera que el sistema toma control del computador colocando en el stack el contenido de los registros CS e IP. El programa suspendido vuelve a activarse cuando termina la ejecución de la interrupción y son restablecidos los registros salvados.

Existen dos razones para ejecutar una interrupción: (1) intencionalmente como petición para la entrada o salida de datos de un dispositivo, y (2) un error serio y no intencional, como sobreflujo (overflow) o división por cero. El operando de una interrupción indica cuál es la rutina a activar.

EJEMPLO

Este ejemplo está desarrollado en lenguaje ensamblador que usa servicios o funciones de MS-DOS (*system calls*) para imprimir el mensaje **Hola mundo!!** en pantalla.

; HOLA.ASM

; Programa clasico de ejemplo. Despliega una leyenda en pantalla.

STACK SEGMENT STACK ; Segmento de pila

DW 64 DUP (?) ; Define espacio en la pila

STACK ENDS

DATA SEGMENT ; Segmento de datos

SALUDO DB "Hola mundo!!",13,10,"$" ; Cadena

DATA ENDS

CODE SEGMENT ; Segmento de Codigo

ASSUME CS:CODE, DS:DATA, SS:STACK

INICIO: ; Punto de entrada al programa

MOV AX,DATA ; Pone dirección en AX

MOV DS,AX ; Pone la dirección en los registros

MOV DX,OFFSET SALUDO ; Obtiene dirección del mensaje

MOV AH,09H ; Función: Visualizar cadena

INT 21H ; Servicio: Funciones alto nivel DOS

MOV AH,4CH ; Función: Terminar

INT 21H

CODE ENDS

END INICIO ; Marca fin y define INICIO

Descripción del programa

1.-Las declaraciones **SEGMENT** y **ENDS** definen los segmentos a usar.

2.-La variable **SALUDO** en el segmento **DATA**, define la cadena a ser desplegada. El signo de dólares al final de la cadena (denominado *centinela*) es requerido por la función de visualización de la cadena de MS-DOS. La cadena incluye los códigos para *carriage-return* y *line-feed*.

3.-La etiqueta **START** en el segmento de código marca el inicio de las instrucciones del programa.

4.-La declaración **DW** en el segmento de pila define el espacio para ser usado por el stack del programa.

5.- La declaración **ASSUME** indica que registros de segmento se asociarán con las etiquetas declaradas en las definiciones de segmentos.

6.-Las primeras dos instrucciones cargan la dirección del segmento de datos en el registro **DS**.

7.-Las últimas dos instrucciones del segmento **CODE** usa la función **4CH** de MS-DOS para regresar el control al sistema operativo. Existen muchas otras formas de hacer esto, pero ésta es la más recomendada.

8.-La directiva **END** indica el final del código fuente y especifica a **START** como punto de arranque.

# Conclusiones

El lenguaje ensamblador es aquel en el que cada enunciado representa una instrucción en el computador. Permite poner nombres simbólicos a las instrucciones y direcciones (ADD; DIV, etc) en vez de tener que recordar el nombre que usa la maquina. Luego del desarrollo, se da a lugar al procedimiento de ensamble, enlace y ejecución.

El ensamblaje convierte un listado de código fuente en código objeto. El enlace y ejecución consiste en:

* Encontrar los módulos a enlazar
* Constituir el módulo de carga asignado a lo segmentos
* Rellenar todos los desplazamientos de segmento no determinados
* Rellenar todas las direcciones de los segmentos
* Cargar el programa

Hay tres tipos de ensambladores:

* Básicos (muy bajo nivel)
* Modulares (se encargan de definir e invocar macro instrucciones)
* Modulares de 32 bits (se encargan de las estructuras de programación complejas)

Compatibilidad

El lenguaje máquina de un procesador no será compatible a menos que se acuerden ciertos estándares para su diseño, ya sea entre modelos de un mismo fabricante, o entre modelos de diferentes compañías.

Estructura

Compuesta por

* Sintaxis
* Campos de una instrucción
* Pseudoinstruciones

Tiene:

* Data segment (contiene variables globales y estáticas que inicializa el programador
* Code segment (parte de la memoria o fichero objeto que contiene las instrucciones ejecutables)
* Stack segment (señala la dirección del segmento donde esta la pila del programa)
* Directivas (se usan par especificar la organización de memoria, realizar ensamblado y defnir macros)

Irrupciones:

Es la ejecución temporaria de un programa o subrutina

Hay dos tipos: de software y hardware

Tiene varios entornos de programación:

Generalmente aparte de un editor, las fuentes se envían a un ensamblador, crea un fichero especial de listado de variables y procedimientos y, por último, el LINKER toma los ficheros objeto y cera el fichero ejecutable final.

Formato de ensamblador

Creado a partir de un archivo fuente de código ensamblador, la sintaxis se forma de la manera:

**[***nombre***]** *mnemónico* **[***operandos***] [*;****comentarios***]**

# Actividades

Responda la(s) respuesta(s) correcta(s)

1. Las rutinas de run-time son…
   1. Tareas ‘batch’ del procesador
   2. Objetos fuente
   3. Información para que el programa pueda ser cargado a memoria y ejecutado
   4. Debuggers en tiempo real
2. De las irrupciones resultan…
   1. Programas borrados
   2. Programas enlazados
   3. Registros guardados
   4. Registros suspendidos
3. El proceso de ensamblado consiste en…
   1. Rellenar todas las direcciones de los segmentos
   2. Especificar las directivas de memoria
   3. Constituir el módulo de carga asignado a lo segmentos
   4. Encontrar los módulos a enlazar
   5. Rellenar todos los desplazamientos de segmento no determinados
   6. Cargar el programa
   7. Todas las anteriores
4. El enlazador o linker produce la salida de…
   1. Módulos
   2. Código fuente
   3. Directivas
   4. Código ejecutable
5. Seleccione las opciones que corresponden a tipos de ensambladores
   1. Microensambladores
   2. Basicos
   3. Desensambladores
   4. Modulares
   5. Macroensambladores
   6. Modulares de 32 bits
   7. Segmentarios
6. El formato correcto del ensamblador es...
   1. [nombre] *mnemónico* [comentarios] [;*operandos*]
   2. [operandos] *mnemónico* [comentarios] [;*nombre*]
   3. [nombre] *mnemónico* [operandos] [;*comentarios*]
   4. [mnemónico] *nombre* [comentarios] [;*operandos*]
7. Un interprete...
   1. Procesa todos los enunciados y luego los traduce
   2. Busca y ejecuta el equivalente al enunciado recibido a medida que procesa cada uno
   3. Envía instrucciones directamente al procesador
   4. Genera un código objeto
8. Las ventajas de programar en ensamblador son
   1. Mayor flexibilidad para utilizar recursos
   2. Portabilidad
   3. Eficiencia
   4. Facilidad de programación
   5. Acceso directo al microprocesador
   6. Disminución de errores
   7. Mejora en la calidad del código
9. La estructura del lenguaje esta compuesta por…
   1. Irrupciones
   2. Campos de una instrucción
   3. Pseudoinstruciones
   4. System calls
   5. Sintaxis
   6. Proceso de liga
   7. Ejecución del programa
10. Una macro dentro del lenguaje ensamblador es...
    1. Una instrucción que contiene varias
    2. Una pseudo-instrucción que define un conjunto de instrucciones asociadas a un nombre simbólico
    3. Un conjunto de módulos
    4. una pseudo instrucción que llama a otros programas

# Bibliografía

* Organización.de.Computadoras-Tanenbaum
* Arquitectura.de.Computadoras-Patricia.Quiroga
* <http://homepage.mac.com/eravila>
* <http://maven.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html>
* La.PC.por.Dentro-Ginzburg